19th ICCRTS
Conceptual Design of Targeted Scrum: Applying Mission Command to Agile Software Development

David P. Harvie
Department of Electrical Engineering & Computer Science
University of Kansas
1520 West 15th Street, Lawrence, KS 66045-7608

Dr. Arvin Agah
Department of Electrical Engineering & Computer Science
University of Kansas
1520 West 15th Street, Lawrence, KS 66045-7608

Abstract

Software engineering and mission command are two separate but similar fields. Both are instances of complex problem solving in environments with ever changing requirements. Also, they have followed similar paths from using industrial age decomposition to deal with large problems to striving to be more agile and resilient. In fact, an entire subset of current software engineering techniques is known as agile methods. Our hypothesis is that modifications to an agile software development methodology (Scrum) based on inspirations from mission command can improve the software engineering process in terms of the planning, prioritizing, and communication of software requirements and progress, as well as improve the overall product. Targeted Scrum is a modification of Scrum based on three inspirations from Mission Command: End State, Line of Effort, and Targeting. Those inspirations led to the creation of the Product Design Meeting and modifications of some current Scrum meetings and artifacts. We propose to test this hypothesis during a semester long graduate level software engineering class. The students will develop software using both methodologies. We will then assess how well these two methodologies assist the software development teams in planning and developing the software architecture, prioritizing requirements, and communicating progress.

1 Introduction

1.1 Similarities between Software Engineering and Mission Command

Why would military command and control (C2), and specifically mission command, be a suitable field to draw inspiration from for software engineering? Military C2 attempts to bring order and

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JUN 2014**	2. REPORT TYPE	3. DATES COVERED **00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE **Conceptual Design of Targeted Scrum: Applying Mission Command to Agile Software Development**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Kansas,Department of Electrical Engineering & Computer Science,1520 West 15th Street,Lawrence,KS,66045-7608**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Presented at the 18th International Command & Control Research & Technology Symposium (ICCRTS) held 16-19 June, 2014 in Alexandria, VA. U.S. Government or Federal Rights License

14. ABSTRACT
Software engineering and mission command are two separate but similar fields. Both are instances of complex problem solving in environments with ever changing requirements. Also they have followed similar paths from using industrial age decomposition to deal with large problems to striving to be more agile and resilient. In fact, an entire subset of current software engineering techniques is known as agile methods. Our hypothesis is that modifications to an agile software development methodology (Scrum) based on inspirations from mission command can improve the software engineering process in terms of the planning, prioritizing and communication of software requirements and progress, as well as improve the overall product. Targeted Scrum is a modification of Scrum based on three inspirations from Mission Command: End State, Line of Effort, and Targeting. Those inspirations led to the creation of the Product Design Meeting and modifications of some current Scrum meetings and artifacts. We propose to test this hypothesis during a semester long graduate level software engineering class. The students will develop software using both methodologies. We will then assess how well these two methodologies assist the software development teams in planning and developing the software architecture, prioritizing requirements, and communicating progress.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**52**	

resolution to military operations which by their nature are chaotic, complex human endeavors that are constantly changing (U.S. Department of the Army [U.S. Army], 2012a). Due to their constantly changing natures, software engineering and military C2 can both be viewed as instances of complex problem solving.

Military C2 has evolved in a similar manner to software engineering. For example, military C2 and software engineering have used industrial age problem decomposition as a means to solve complex problems. The problem is analyzed, task and requirements are derived, and a scheme is developed that implements those tasks and requirements. In the Army, this is embodied by the Military Decision Making Process (MDMP)(U.S. Department of the Army [U.S. Army], 2012b). Comparing MDMP to the Waterfall Model (see Table 1), it becomes apparent that there are strong similarities between traditional military problem solving and software development.

Step	Military Decision Making Process	Waterfall Model
1	Receipt of Mission	System Requirements
2	Mission Analysis	Software Requirements
3	Course of Action (COA) Development	Analysis
4	COA Analysis (War Game)	Program Design
5	COA Comparison	Coding
6	COA Approval	Testing
7	Orders Production	Operations

Table 1: Military Decision Making Process (U.S. Army, 2012b) and Waterfall Model Comparison (Royce, 1970).

Assessment is another area of similar concern to software engineering and mission command. Software developers assess their products via validation and verification. Validation is concerned with whether the delivered software meets the customer's requirements. Verification is concerned with whether a specified piece of software functions as it is designed to. Barry Boehm succintly summarized validation as "Are we building the right product?" and verification as "Are we building the product right?" (Pressman, 2005). Military actions on a small scale (or tactical level) are rather straight forward to assess. Did the infantry battalion secure the bridge as ordered to? On the larger scales (such as operational and strategic levels), it is not so easy to assess the effects of operations. The conditions necessary to achieve a desired military end state at those levels (e.g., supporting governance in a host nation) are much more nebulous. It is therefore necessary to determine Measures of Effectiveness (MOEs) and Measures of Performance (MOPs) in order to assess progress in a military operation or campaign. MOEs, similar to validation, ask, "Are we doing the right things?" Likewise, MOPs ask, "Are we doing things right?" (U.S. Department of Defense[DOD], 2011a). These similarities of validation and verification to MOEs and MOPs suggest that the software development and military communities have lessons they can learn from each other. These examples are part of the justification for looking for inspiration from mission command.

1.2 Research Hypothesis

Currently, software engineering and military C2 (as expressed by mission command) are striving to be more agile and resilient to changing environments. Neither field has the panacea for complex problem solving. However, the military has developed some good techniques for responding to change due to over 11 years of continuous combat. Our hypothesis is that modifications to agile software development based on inspirations from mission command can improve the software engineering process in terms of the planning, prioritizing, and communication of software requirements and progress, as well as improve the overall software product. We propose to test this hypothesis by developing two software projects, one using the traditional agile software development methodology, Scrum, and the other using the proposed agile software methodology, called Targeted Scrum. Our objective is to test our hypothesis for agile software development specifically to Scrum with the intent that the positive results may lead to applying these inspirations to potentially agile software development methodologies as a whole.

However, modifying an agile software development process is not without its tradeoffs. With a military inspiration, there will tend to be a little more formalism and more meetings which may make the process feel a little less agile. However, we hypothesize that the benefits of improved planning prioritizing, and communication of software requirements will offset any perceived loss of agility.

1.3 Approach Summary

We propose to use inspiration from mission command to improve a specific agile software development method, namely, Scrum. Two specific weaknesses with Scrum are a lack of initial planning and a lack of an overall architecture. This proposal addresses the first weakness with the addition of a Product Design Meeting at the onset of the Scrum process. This meeting is subsequently repeated in Scrum to assess the validity and to make necessary changes to the product's design. This proposal addresses the second weakness with the addition of two artifacts, namely, the product's end state and lines of effort (LOEs). Both of these artifacts are derived from mission command. They will be used to establish and communicate the product's overall architecture. The additional meeting and artifacts should assist with the initial planning with Scrum and contribute to a more stable product framework.

2 Background of Scrum

2.1 Development of Scrum

An agile software development methodology is defined as "an approach based on iterative development, frequent inspection and adaptation, and incremental deliveries" (ISO, IEC, & IEEE, 2012). Several software development approaches such as Extreme Programming (XP), Crystal, Scrum, Feature Driven Development, and Dynamic Systems Development Method (DSDM), fall under the umbrella of agile software development. XP and Scrum are two of the most popular agile software development methodologies.

XP was developed by Kent Beck in the late 1990s. The costs associated with implementing changes in a traditional software development methodology followed an exponential curve. The motivating principle behind XP was to flatten the cost of implementing changes in software development. XP adhered to five principles to achieve this stabilized cost of change: rapid feedback, assume simplicity, incremental change, embrace change, and quality work (Beck, 1999).

Scrum is a management process, first implemented by Ken Schwaber in 1996, that can be incorporated with existing engineering processes such as software engineering. It is associated with Agile Software Development and has been used in conjunction with Extreme Programming. Scrum is based on the premise that complex activities, such as software development, are impossible to fully predict (Schwaber & Beedle, 2002). Change, therefore, is unavoidable and must be addressed. Scrum deals with change by building software in increments versus an all-at-once approach. Scrum also makes continuous assessment throughout the building process. Finally, Scrum reviews the finished increment and makes the appropriate changes for the next increment building process.

Scrum begins with the Scrum Team, a small, cross-functional team consisting of approximately seven people. The team is autonomous and self-organizing. The management representative on the team is the Scrum Master. The primary responsibilities of the Scrum Master are to remove obstacles to the team and to ensure Scrum practices are followed. The team operates best in an open work environment with white boards in order to facilitate communication and team interaction (Schwaber & Beedle, 2002).

Each project has a Product Backlog, which is a prioritized list of all requirements to achieve. It is an ever-changing document based on changing requirements, changing understanding of the problem by both the Product Owner and the Scrum Team, and changing environments. Only the Product Owner has the authority to prioritize the Product Backlog, though many others can give input into what should be placed on the Product Backlog. Initially, the Product Backlog may be very small as both the Product Owner and Scrum Team brainstorm to identify all the requirements that belong in the Product Backlog. The minimal Product Backlog has sufficient requirements to fill a Sprint Backlog (Schwaber & Beedle, 2002).

The management of the Product Backlog can sometimes be referred to as "grooming." Rubin defines grooming as the process of managing the Product Backlog by creating and refining Product Backlog Items (PBIs), estimating the amount of work to accomplish a PBI, and prioritizing the PBIs in the Product Backlog (Rubin, 2012). The grooming process keeps the Product Backlog as a relevant planning document and prevents it from becoming a haphazard collection of tasks that need to be done.

Scrum breaks a project into increments called Sprints. A Sprint is a single, usually 30-day, iteration during which the Scrum Team adds new functionality to the Product. The Sprint begins with a Sprint Planning Meeting. During the first part of the Sprint Planning Meeting, the Product Owner, Scrum Master, and Scrum team determine which Product Backlog Items should and can be developed during the upcoming Sprint. This portion of the Product Backlog is translated into an objective called the Sprint Goal. During the second part of the Sprint Planning Meeting, the Scrum Team meets internally. Using the Sprint Goal and selected Product Backlog items, the Scrum Team determines the specifics of how it will work as a team to achieve the specific Sprint Goal. These work specifics are then placed into the Sprint Backlog. The Scrum Team can change the specifics of the Sprint Backlog within the Sprint, but they cannot change the Sprint Goal. Success of the Sprint is dependent on whether the Sprint Goal has been achieved to the satisfaction of the Product Owner and Scrum Team (Schwaber & Beedle, 2002). The short time span of Sprints to 30 days or less facilitates greater stability and predictability as most project requirements will likely remain stable during those short time periods.

During the Sprint, the Scrum Master holds a daily 15-minute meeting with the Scrum Team. The purpose of this meeting, called the Daily Scrum, is to review progress. Each member of the team answers three questions:

- "What has been accomplished since the last meeting?"
- "What will be done before the next meeting?"
- "What obstacles are in the way?" (Schwaber & Sutherland, 2011)

This entire traditional Scrum framework can be viewed as a whole in Figure 1.

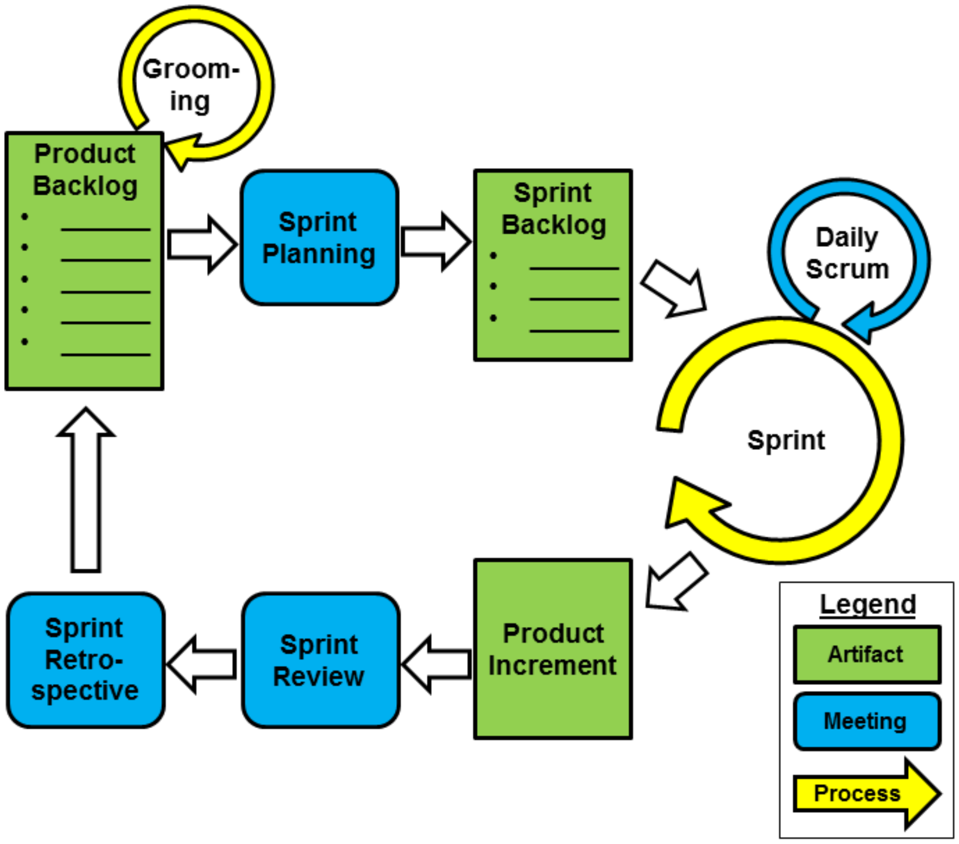


Figure 1: Traditional Scrum Framework (Rubin, 2012).

For larger software projects, there are multiple Scrum teams who meet daily. After those Daily Scrums, the Scrum Masters meet together with the Product Owner in a "Scrum of Scrums" to review progress (Schwaber & Beedle, 2002).

The end result of the Sprint is a deliverable product increment. The Product Owner, Scrum Master, and Scrum Team then meet for a 4-hour Sprint Review. The product increment is delivered, and the Scrum Team presents an assessment of what they were able to accomplish during the Sprint. This assessment of the work accomplished is compared to the Sprint Goal and Product Backlog with the purpose of updating the Product Backlog prior to the start of the next Sprint Planning meeting (Schwaber & Beedle, 2002). The Sprint Review is followed by the Sprint Retrospective which is an internal Scrum Team meeting where the team assesses its work and its processes. The purpose of the Retrospective is to improve the team's processes, and ultimately their work, prior to the next Sprint (Schwaber & Sutherland, 2012).

The theory behind Scrum is empiricism. Schwaber and Sutherland (2011) define empiricism as making decisions based on experience and what is currently known at the time. A critical part of this empiricism is frequent inspections to determine if the process is on track and subsequently making the necessary adaptations to correct for variances. In some ways, this Scrum cycle is similar to the Observe Orient Decide Act (OODA) loop proposed by Colonel John Boyd. The commander in the OODA loop uses his or her experience and awareness of the enemy and situation to focus on a specific area much like empiricism. Also, the OODA has multiple feedback loops going back to the Observe component that enables the commander to understand changes in the situation and make appropriate adjustments (Coram, 2002). These feedback loops are similar to the frequent inspections in Scrum. However, there are some key differences between Scrum and the OODA loop. Scrum was designed to build and manufacture code. The OODA loop was designed by a fighter pilot to defeat an adversary by understanding and adapting to the environment more quickly than the adversary could (Coram, 2002). Also, Scrum is necessary designed for team of about seven people, while the OODA loop can be successfully applied by individuals or organizations.

2.2 Analysis of Scrum - Strengths & Weaknesses

The main strengths of the Scrum are its iterative process and continuous feedback. Li *et al.* (2010) surveyed a company that transitioned from a plan driven software methodology to Scrum. Their survey found that the continuous daily feedback in addition to the feedback received during the sprint retrospective meetings resulted in a greater focus on software quality. Also, the daily scrum meetings fostered developer communication and interaction resulting in more team synergy. Sutherland *et al.* (2007), likewise saw similar positive aspects of Scrum from his experience with Systematic. Constant communication with the client and iterative software releases directly led to the early discovery of defects and technological issues. This early discovery reduced costs and scheduling issues associated with discovering defects later in the process. Customer satisfaction also increased because of the transparent process involving feedback and iterative releases.

Much of the Scrum's focus is on iteration, assessment, and client feedback. However, there is much less effort on the initial identification of requirements at the beginning of the project. Hochmüller & Mittermeir (2008) criticize agile methods for their lack of initial design. Requirements may unnecessarily change in the project as both the software client and software developers learn what requirements are necessary. In addition, for many projects there are already established architecture and schema that should be followed in order to achieve success. Overhage & Schlauderer (2012) cite the lack of initial planning that makes it more difficult for Scrum teams to begin a project. The team must execute a couple of sprints before the overall architecture and long-range planning begins to be established.

Another weakness of Scrum can be the Product Owner (i.e., software client representative). This person can be a single point of failure depending on his or her knowledge of what the system is supposed to do. The client may not dedicate the most knowledgeable person to the development team because that person's expertise is also needed by the client. This results in decreased responsiveness as the Product Owner has to reach back to find answers to the team's questions

(Hochmüller, 2011)(Hoda et al., 2010).

A third challenge in Scrum, closely related to the first, is the lack of focus on design. Drury *et al.* (2012) report that many of the decisions made during Scrum planning were more tactical in nature (i.e., which tasks to accomplish and how to accomplish them) versus strategic (i.e., which future tasks are necessary). The planning focus is persistently on what requirements can be fulfilled during the next Sprint. There is very little discussion on the overall design of the system because the design is considered to be "emergent." The assumption is that the design will naturally evolve as the system grows.

2.3 Other Enhancements of Scrum

In order to capitalize on the strengths of Scrum while mitigating its weaknesses, several hybrid methods have been proposed. Sutherland *et al.* (2007) proposed institutionalizing agile methods, such as Scrum, using the 12 Generic Processes (GPs) associated with Capability Maturity Model Integrated (CMMI) Levels 2 and 3. The essence of this plan is to insert agile processes into an already established CMMI organization with the intent for the organization to be both disciplined and adaptable. The authors' work was based on their experience at Systematic, a software systems company that had already achieved CMMI Level 5. The company had experimented with Lean Software Development, and the company decided to pilot two large and two small projects using Scrum and early testing. This approach does bring institutional stability but it also appears to hamper the adaptability and flexibility that agile methods have.

Lina & Dan (2012) also proposed combining CMMI and Scrum. The authors conceptualize CMMI as a high level abstraction of project management focused on the organization and Scrum as a lower level abstraction focused on the software development. The two systems can then complement each other. CMMI can be used to implement risk management and mitigation, configuration management, and product quality assurance since these areas are not explicitly addressed in Scrum. The authors did not cite any empirical data based on their proposal in this paper.

Rong *et al.* (2010) proposed merging Scrum with Personal Software Process (PSP). Each sprint, redefined as an iteration, would consist of five steps: Launch, Plan, Requirements and Design, Construction, and Postmortem. In the Requirements and Design step, the development team not only looks to the design of the current iteration but also to how that iteration fits in with higher level design. This deliberate focus on both higher and lower level design is a positive aspect of this hybrid approach. However, this focus appears only within the iterations. There is not an initial development of requirements to establish that higher level design. The authors implemented their proposal with a team of five members to add more features to an existing web based application. One can assume that the five members were students because the application was to be used by students and no company name or description is given.

Maranzato *et al.* (2012) described a process called "Mega Framework" used to manage multiple Scrum teams working on a project. The project was too large for a single team to accomplish so they created multiple Scrum teams. The teams were organized according to software features in order to avoid the complexity of managing coupling between software components. They also created a "Mega Backlog" that consisted of higher level themes with enough complexity to communicate with the software client. The individual Scrum (feature) teams maintained the Backlogs with more specific details. They also grew additional teams by splitting a team and adding new personnel to each team. This allowed tacit knowledge in both teams to be spread from the veteran members to the newer members. Finally, they also created additional meetings (Mega Planning, Mega Standup, and Mega Retrospectives) that facilitated synchronization and knowledge sharing among the teams. The authors based their proposal on their experience in the Research and Development department of Universo Online, the largest internet service provider in Brazil.

Miranda & Borque (2010) proposed using a line of balance (LOB) as a tool to capture and communicate progress in Scrum. The LOB is defined is a chart that shows the number of items that actually pass a specific control versus the number of items that were scheduled to pass that same control point. The LOB would give a more accurate assessment of progress in Scrum than the traditional burn down charts. A burn down chart is a graphical representation of the tasks remaining to be accomplished in either a release or iteration. Progress is shown as the amount of remaining work "burns down" to nothing left. They used LOB for the various stages of software development: starting, designing, testing, acceptance, and release. The LOB enabled the authors to identify which stages had bottlenecks which could result in a reallocation of focus and resources. While the LOB does provide a better assessment of progress versus burn down charts, it still does not communicate when important milestones are being met or not. This work was based off experience of the lead author implementing LOBs at a large telecommunication supplier.

3 Targeted Scrum

3.1 Approach

Targeted Scrum attempts to provide the initial planning and overall design architecture that are missing in traditional Scrum. Developing code for two or three Sprint iterations before an overall architecture begins to emerge will naturally generate bad code, also called technical debt. This technical debt of defects and bugs will have to be repaid by further code fixes (Allman, 2012). There is always some level of technical debt in any software project, but it is inefficient to generate unnecessary technical debt due solely to the fact that no framework yet exists. There needs to be an initial framework of the project even if it is known that this preliminary framework can and will change. Committing intellectual energy at the beginning of the project to establish that initial framework will reduce the amount of technical debt accumulated at the onset of the project. The development of this framework will require the addition of an initial planning meeting prior to the creation of the Product Backlog.

Likewise, the prioritization of the Product Backlog and selection of the Product Backlog Items that are to be a part of each Sprint Backlog needs to be based on the architecture of the program. LOEs (see §1.3 for introduction) can assist in communicating both the architecture and prioritization of the project. The Product Owner and Scrum Master can make intelligent decisions on what needs to be done next by evaluating how completing specific Product Backlog Items will advance progress along the LOEs toward the project's ultimate end state.

Targeted Scrum will enhance traditional Scrum in four ways:

1. The addition of the Product Design Meeting will establish a Product framework that is missing in traditional Scrum;

2. The LOEs and Product Design Meeting will establish a formal mechanism to better groom the Product Backlog;

3. The LOEs will improve Sprint Planning by enabling the Scrum team to select a more appropriate set of Product Backlog Items to be incorporated into the next Sprint; and,

4. The LOEs will provide a better mechanism to communicate the Product's progress to stakeholders that are both internal and external to the Scrum team.

The targeted Scrum framework can be viewed as a whole in Figure 2.

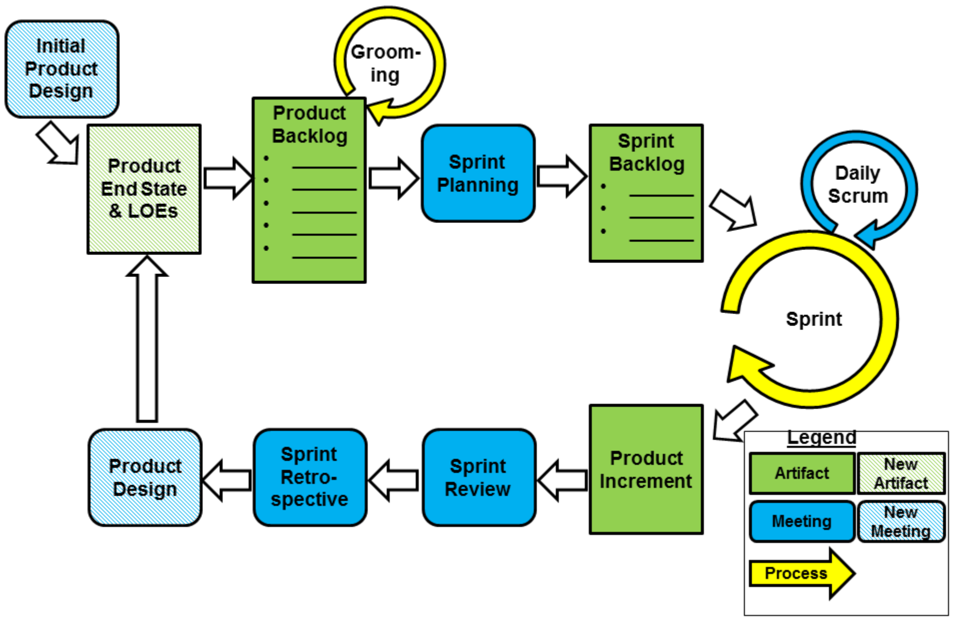


Figure 2: Targeted Scrum Framework.

Targeted Scrum is a modification of traditional Scrum based on inspirations from Mission Command. The three inspirations of Mission Command are End State, LOE, and Targeting. Figure 3 shows the mapping of those inspirations to the four areas of modification of traditional Scrum.

3.2 Product Design Meeting

Traditional Scrum begins with the creation of the Product Backlog based on the requirements of the Product. There is no formal mechanism of how to derive and prioritize these requirements and subsequently incorporate them into the Product Backlog. The derivation and prioritization of requirements are significant and they can have serious ramifications on Product development if done improperly.

The first step in Targeted Scrum is the addition of a Product Design Meeting. The initial Product Design Meeting will take place prior to the creation of the Product Backlog. The first task of this Product Design Meeting is for the Scrum Master and the Product Owner (also referred to as the software client) to determine the end state for the project. What does the client's environment look like with the fully developed software? As stated in JP 5-0 Joint Planning, "effective planning cannot occur without a clear understanding of the end state" (U.S. Department of Defense[DOD], 2011b). This end state is further detailed through the use of supporting objectives that the Product will accomplish.

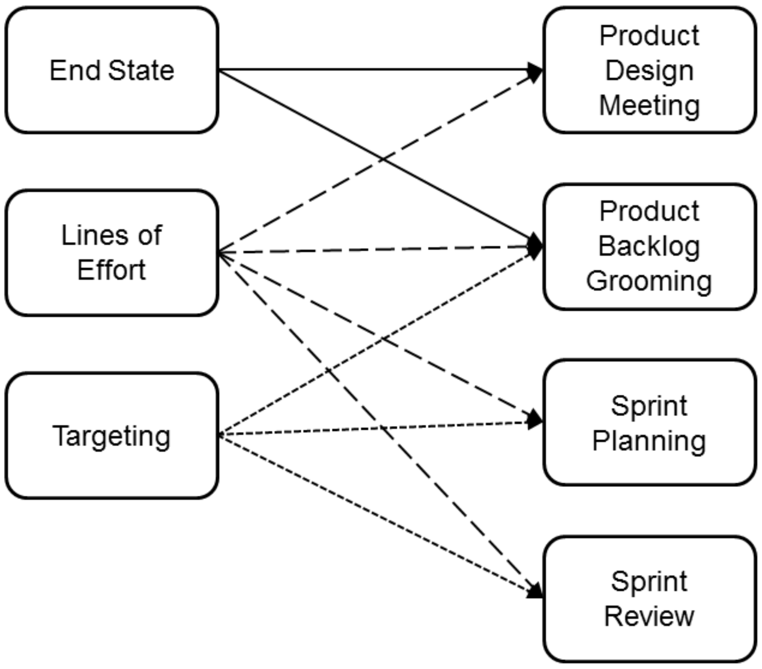


Figure 3: Mapping between Mission Command Inspirations and Traditional Scrum Modifications.

The second task of the Product Design Meeting is to determine the critical software features necessary to achieve the end state and its objectives. The Product Owner and Scrum Master turn these critical features into LOEs. Each LOE has an end state, which should nest within the end state and objectives of the project. In order to reach the end state for the specific LOE, the team develops goals and milestones that are measurable and represent significant progress towards the end state. In larger software development teams, these LOEs can be the basis to form Scrum teams around, in case the project is large enough to require multiple Scrum teams. Likewise, the LOEs can be an effective tool to communicate progress and synchronize efforts across the Scrum teams.

The initial Product Design Meeting should last no more than two hours. The agenda of that initial meeting is outlined in Table 2.

The following is a partial example of an initial Product Design Meeting. The software client wants a web-based personnel management system for her organization. The first step is to identify the end state for the product. In this case, the client wants a secure, efficient, and easily adaptable personnel management system for an organization of 50-200 people that can be accessed and modified via a Web browser. During the Scrum Master and Product Owner dialogue, several product features are identified including security, Web-based interface, and personnel data base. The personnel data base is further identified as a critical feature. The next step is to identify the end state for the data base. The end state is articulated as a fully integrated data base with access controls. This critical feature with end state is then transformed into an LOE. The final step for this particular LOE is to identify measurable milestones such as identifying all schemas, entities, and relationships, populating the built data base, and implementing access controls. This LOE is shown in Figure 4.

Subsequent Product Design Meetings take place after every two to three Sprints. These meetings should be limited to one hour. The purpose of these subsequent Product Design Meetings is

Step	Question and/or Action
1	What is the goal/end state of the delivered product?
2	What are the features necessary to implement in order to reach the product end state?
3	Which of the identified features are absolutely critical to the product's success?
4	For each critical feature, what is the end state for that critical feature?
5	Transform each critical feature into a line of effort (LOE) with associated end state.
6	For each LOE, identify measurable milestones that mark progress towards the end state.

Table 2: Agenda for the Initial Product Design Meeting.

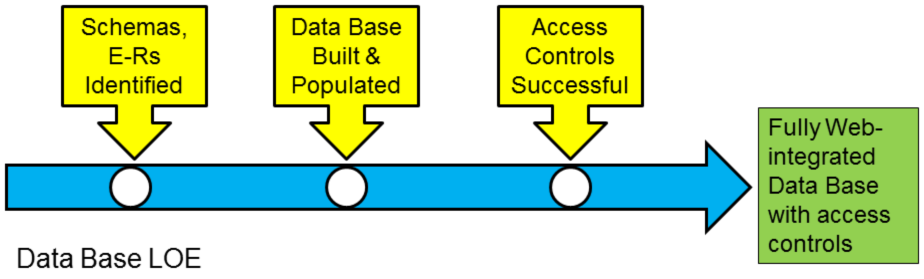

Schemas, E-Rs Identified
Data Base Built & Populated
Access Controls Successful
Fully Web-integrated Data Base with access controls
Data Base LOE

Figure 4: Example of Data Base LOE.

to assess progress along the LOEs and progress towards to the overall end state. This meeting also assesses whether the product end state, objectives, and LOEs are still valid and relevant in their current forms or do they need to be modified. These assessments reflect the reality that requirements, program knowledge, and environments can and will change. The reason for holding this meeting after multiple Sprints is to allow some continuity between Sprints. The agenda for these meetings is shown in Table 3.

3.3 Grooming the Product Backlog

In traditional Scrum, grooming is a part-time, ongoing dialogue between the Product Owner and the Software Development Team. The Development Team is considered the senior partner in this conversation due to the assumption that they have more knowledge of the domain and their capabilities than the client. The client influences the Scrum team by communicating the tradeoffs associated with Product Backlog decisions (Schwaber & Sutherland, 2011). One issue is: What basis is used for prioritizing and refining the Product Backlog Items? A second issue is: When is the most appropriate time for this dialogue to take place?

Step	Question and/or Action
1	Review the goal/end state of the delivered product. Is it still valid?
2	Review each LOE with its associated end state and milestones. Is each LOE still valid? Does any LOE need to be modified, added or deleted?
3	Review the progress and prioritization of the LOEs. Is the current prioritization of LOEs still valid?
4	Publish any necessary updates to the end state and/or LOEs.

Table 3: Agenda for the Subsequent Product Design Meeting.

In targeted Scrum, Product Backlog grooming is an intentional conversation that is held after the Product Design Meetings. Formalizing the time and place for the Product Backlog grooming will ensure that this essential dialogue takes place on a routine basis. Also, the timing of the Product Design Meetings in relationship to subsequent Sprints will help facilitate the Scrum team's response to changes that result from the grooming process.

In targeting doctrine, an individual target's true importance is relative to its relationship to other targets (U.S. Department of Defense[DOD], 2013). Applying that concept to grooming, a Product Backlog Item's importance can only be measured with respect to the other Product Backlog Items. The creation and grooming of the Product Backlog can now be done against the backdrop of the Product end state, objectives, and LOEs. Product Backlog Items are identified and prioritized based on how they move the product along its LOEs towards it objectives and end state. Changes in the Product end state, objectives, and/or LOEs will necessitate changes in the content and prioritization of the Product Backlog.

The LOEs provide the mechanism to formalize this dialogue as they are visual representations of the software client's priorities and desire end state. The LOEs themselves can be prioritized in some order to further communicate a shared understanding of both the desired product and the steps necessary to build that product.

3.4 Sprint Planning

In traditional Scrum, Sprint Planning is a dialogue between the Product Owner, Scrum Master, and Scrum team to determine which Product Backlog Items are to be implemented into the next Sprint. The selected items are then translated into a Sprint Goal. Part of this selection process is to determine which items are feasible to implement, given the duration of an individual Sprint. However, the selection of Product Backlog Items being used as the basis to determine the Sprint Goal seems counterintuitive. The Sprint Goal should be determined first, and then the Product Backlog Items to be implemented should be derived from that Sprint Goal.

In targeted Scrum, each Sprint Planning meeting begins with a review of the Product end state, objectives, and the current progress along the LOEs. The Scrum team can then determine the most appropriate Sprint Goal to progress along the LOEs. The Scrum team now can develop the Sprint

Backlog for the upcoming Sprint based on that Sprint Goal. The chosen Product Backlog Items are measured against how they will help facilitate movement along the LOEs. The program end state, objectives, and LOEs will assist the Scrum team in adhering to the overall program design when planning a Sprint. All other agenda items in the Spring Planning meeting remain unchanged.

3.5 Sprint Review

In traditional Scrum, the 4-hour Sprint Review is the meeting at which the Product increment is delivered. This is also the opportunity for the Scrum team to brief what they were able to accomplish during the Sprint in relation to the Sprint Goal and overall Product Backlog. Progress is measured as completed Product Backlog Items are fulfilled. One issue with using completed Product Backlog Items to measure progress is that the Scrum team and/or Product Owner may have identified additional requirements that need to be added to the Product Backlog. Thus, it is possible for the number of unfulfilled Product Backlog Items to constantly shrink and grow based on identifying new requirements.

An addition to the Sprint Review meeting in targeted Scrum is to visually update progress along the LOEs. An example of this is shown in Figure 5. This updated progress will assist the Scrum team to prepare for the next Sprint Planning meeting by communicating possible areas that they need to address next. The visualization of progress (or non-progress) makes the dialogue of the Spring Review more meaningful between the Product Owner and the Scrum team. Is the project progressing as anticipated? Are there concerns that need to be addressed quickly? Likewise, these updated LOEs help to succintly communicate the product's status to the Product Owner. This is especially important if the the Product Owner is a company representative who must then relay that progress to superiors who are not directly part of the Scrum process.

The LOEs and their associated milestones are considered valid at this point. The only issue to consider is the amount of progress made along the LOEs. This constrained focus keeps the Sprint Review on track. The Sprint process may have uncovered problems with the validity of the Product end state, objectives, and LOEs. However, those concerns should be addressed in subsequent Product Design Meetings.

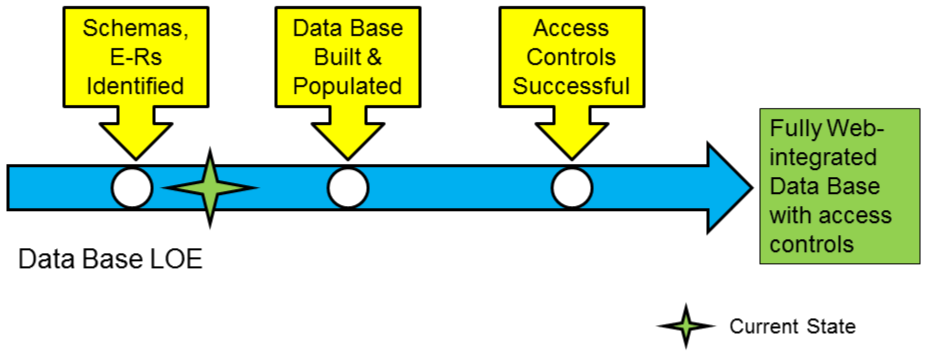

Schemas, E-Rs Identified
Data Base Built & Populated
Access Controls Successful
Fully Web-integrated Data Base with access controls
Data Base LOE
Current State

Figure 5: Example of Updated Data Base LOE.

4 Proposed Experiment

4.1 Protocols

We propose to test the research hypothesis during a semester long software engineering course consisting of both graduate and upper level undergraduate students in Computer Science and Computer Engineering. The class will be divided into teams of 3–4 students per team. Given the time restraints of a one-semester course, there will be two assigned software development projects. The Sprint duration will be set to two weeks, and a single project will consist of two Sprints. For the first project, one-half of the teams will develop the software using Traditional Scrum while the remaining half will use Targeted Scrum. For the second project, the teams will switch development methodologies. Students going into the second project will have the benefit of learning from the first project. The purpose assigning one-half of the teams to using Traditional Scrum while the other half is assigned Targeted Scrum is to negate the effect of students performing better on the second project from biasing the results in favor of whatever methodology is used during the second project. The instructor will serve as the customer for all teams in order to set and communicate a common set of objectives. The programming language for all projects will be standardized to Java (Oracle, 2013). Also, the projects will be developed utilizing the Eclipse Integrated Development Environment (IDE) in order to take advantage to plugins developed for that IDE (The Eclipse Foundation, 2013).

The archiving of artifacts related to the software development process will be critical to the accurate evaluation of both Traditional Scrum and Targeted Scrum. Artifacts, such as Product Backlogs, Sprint Goals, and customer questionnaires will be immediately time stamped and saved.

4.2 Assessing the Processes

The assessments of the Traditional Scrum and Targeted Scrum processes will be done using two sets of evaluators. The first set of evaluators are the students. They will be given an initial survey to measure their knowledge and experience with software engineering, particularly agile software engineering. After each project, the students will be surveyed to measure their assessment of the particular software development methodology used. Since the students are directly involved in both software development processes, it is essential to capture their assessments.

Some of the student assessments will include:

- How well did the method you used contribute to identifying requirements?
- How well did the method you used contribute to defining the overall architecture?
- How well did the method you used contribute to communicating internally?
- Identify two strengths and two weaknesses of the method you used.

At the end of the semester, the students will be given a final survey in which they evaluate both Traditional Scrum and Targeted Scrum against each other.

The second set of evaluators are two outside observers who will inspect the artifacts developed by both the Traditional Scrum and Targeted Scrum processes. The artifact include, but are certainly not limited to Product Backlogs, Sprint Goals, Sprint Backlogs, and Programming Work Logs.

The purpose of these two evaluators is to give an unbiased assessment of the two processes relying solely on the artifacts produced. These evaluators will conduct their assessments after two software iterations, a Traditional Scrum process and a Targeted Scrum process, so that they can directly compare artifacts against each other.

Some of the outside observer assessments will include:

- Given the artifacts, how well do you understand the Product's goal (or end state)?
- What is your assessment of the Product's design (or architecture)?
- How well does the Product Backlog reflect the software client's priorities, given the Product's goal?
- How well does the selection of Product Backlog Items to be part of a particular Sprint reflect the client's priorities?

The results from the two sets of assessments, student and outside observer, will then be combined and analyzed to yield an overall assessment of the process used for each iteration and the research hypothesis.

4.3 Assessing the Product

The assessments of the delivered software products will be done by both automatic and personal evaluations. The automatic evaluations will be captured using the Metrics 1.3.6 plugin (Sauer, 2013) for the Eclipse IDE. The Metrics plugin will be utilized to capture, at a minimum, the following metrics:

- Afferent Coupling
- Efferent Coupling
- Instability
- Lack of Cohesion of Methods
- McCabe Cyclomatic Complexity

The automated metrics will be used to provide an unbiased and consistent evaluation of the quality of the code for each software development iteration.

The personal evaluations will be conducted by the instructor who will assess the quality of the final delivered software product. The product will be assessed at how well it meets the client's requirements and the overall quality of the software. The metrics and software client evaluations will be used to give an overall assessment of each software product.

Some of the qualitative assessments about the product will include:

- How well does the final delivered product meet your expectations?
- What is your assessment of the quality of the software?
- What is your assessment of the ease of use of the software?
- What is your assessment of the maintainability of the software?

4.4 Start of Experiment

We began the experiment during the Spring 2014 semester at the University of Kansas. We offered the class as an elective course of upper level undergraduate and graduate students in Computer Science. We had 26 undergraduate students, juniors and seniors, enroll in the course. We then allowed the students to self-divide into teams of 3–4 members. For the experiment, we have five teams of four members and two teams of three members. The first project was completed in its entirety prior to Spring Break with the second project beginning after students return from Spring Break.

A key point was to replicate the difficulty and experience of the first project with the second project. This effort included the insertion of a new requirement between the first and second Sprints of each project in order to cause the student to respond to this change. One anecdotal observation of the second project in progress is that all the teams appeared to have benefitted from the experience of coding in Java using the Eclipse IDE from the first project. This anecdotal evidence supports our decision for half of the teams to utilize Traditional Scrum and other half to utilized Targeted Scrum for each project in order to negate the learning effect between projects from biasing our results.

4.5 Closing Discussion

Adding rigor to an agile software development methodology may make it feel more like a plan-driven methodology. However, Targeted Scrum is not a more agile version of the Waterfall Method. The Waterfall Method attempts to identify all requirements at the beginning of the process. On the other hand, Targeted Scrum focuses initially on just the end states and critical features of the product which are later translated into LOEs. By keeping focus on the desired end states and LOEs, Target Scrum allows for a more natural identification and evolution of requirements than the Waterfall Method. Targeted Scrum does require additional overhead as compared to Traditional Scrum. Nonetheless, we hypothesize that the limited additional effort will be offset by more productive efforts.

References

Allman, E. (2012, March). Managing technical debt. *Queue*, *10*(3), 10:10–10:17. Retrieved from `http://doi.acm.org/10.1145/2168796.2168798`

Beck, K. (1999). *eXtreme programming explained: Embrace change*. Boston, MA: Addison-Wesley Publishing Company.

Coram, R. (2002). *Boyd: The fighter pilot who changed the art of war*. Boston, MA: Little, Brown and Company.

Drury, M., Conboy, K., & Power, K. (2012). Obstacles to decision making in agile software development teams. *Journal of Systems and Software*, *85*(6), 1239 - 1254. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0164121212000374` (Special Issue: Agile Development)

Hochmüller, E. (2011). The requirements engineer as a liaison officer in agile software development. In *Proceedings of the 1st workshop on agile requirements engineering* (pp.

2:1–2:4). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2068783.2068785`

Hochmüller, E., & Mittermeir, R. T. (2008). Agile process myths. In *Proceedings of the 2008 international workshop on scrutinizing agile practices or shoot-out at the agile corral* (pp. 5–8). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1370143.1370145`

Hoda, R., Kruchten, P., Noble, J., & Marshall, S. (2010). Agility in context. In *Proceedings of the acm international conference on object oriented programming systems languages and applications* (pp. 74–88). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1869459.1869467`

ISO, IEC, & IEEE. (2012, March). Systems and software engineering – developing user documentation in an agile environment. *ISO/IEC/IEEE 26515*, 1–36.

Li, J., Moe, N. B., & Dybå, T. (2010). Transition from a plan-driven process to scrum: a longitudinal case study on software quality. In *Proceedings of the 2010 acm-ieee international symposium on empirical software engineering and measurement* (pp. 13:1–13:10). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1852786.1852804`

Lina, Z., & Dan, S. (2012, March). Research on combining scrum with cmmi in small and medium organizations. In *Computer science and electronics engineering (iccsee), 2012 international conference on* (Vol. 1, pp. 554–557).

Maranzato, R., Neubert, M., & Herculano, P. (2012, August). Scaling scrum step by step: "the mega framework". In *Agile conference (agile), 2012* (pp. 79–85).

Miranda, E., & Bourque, P. (2010). Agile monitoring using the line of balance. *Journal of Systems and Software*, *83*(7), 1205–1215. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0164121210000294`

Oracle. (2013). *Java.* Retrieved from `http://www.java.com`

Overhage, S., & Schlauderer, S. (2012, January). Investigating the long-term acceptance of agile methodologies: An empirical study of developer perceptions in scrum projects. In *System science (hicss), 2012 45th hawaii international conference on* (pp. 5452–5461).

Pressman, R. S. (2005). *Software engineering: A practioner's approach* (6th ed.). New York, NY: McGraw-Hill.

Rong, G., Shao, D., & Zhang, H. (2010, December). Scrum-psp: Embracing process agility and discipline. In *Software engineering conference (apsec), 2010 17th asia pacific* (pp. 316–325).

Royce, W. W. (1970, August). Managing the development of large software systems. In *Proceedings of IEEE WESCON* (p. 1 - 9). New York, NY: IEEE Computer Society Press.

Rubin, K. S. (2012). *Essential scrum: A practical guide to the most popular agile process.* Upper Saddle River, NJ: Addison-Wesley.

Sauer, F. (2013, April). *Metrics 1.3.6.* Retrieved from `http://metrics.sourceforge.net`

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum.* Upper Saddle River, NJ: Prentice Hall.

Schwaber, K., & Sutherland, J. (2011, October). *The scrum guide, the definitive guide to scrum: The rules of the game.* Retrieved from `http://www.scrum.org/Portals/0/Documents/ScrumGuides/Scrum_Guide.pdf`

Schwaber, K., & Sutherland, J. (2012). *Software in 30 days: How agile managers beat the odds, delight their customers, and leave competitors in the dust.* Hoboken, NJ: John Wiley &

Sons.

Sutherland, J., Jakobsen, C., & Johnson, K. (2007, August). Scrum and cmmi level 5: The magic potion for code warriors. In *Agile conference (agile), 2007* (pp. 272–278).

The Eclipse Foundation. (2013). *Eclipse.* Retrieved from `http://www.eclipse.org`

U.S. Department of Defense. (2011a, August). *Joint publication 3-0 joint operations* [Joint doctrine manual]. Washington, DC: US Government Printing Office.

U.S. Department of Defense. (2011b, August). *Joint publication 5-0 joint operation planning* [Joint doctrine manual]. Washington, DC: US Government Printing Office.

U.S. Department of Defense. (2013, January). *Joint publication 3-60 joint targeting* [Joint doctrine manual]. Washington, DC: US Government Printing Office.

U.S. Department of the Army. (2012a, May). *Army doctrine publication 6-0 mission command* [Army doctrine manual]. Washington, DC: US Government Printing Office.

U.S. Department of the Army. (2012b, May). *Army doctrine reference publication 5-0 the operations process* [Army doctrine manual]. Washington, DC: US Government Printing Office.

THE UNIVERSITY OF KANSAS

Conceptual Design of Targeted Scrum: Applying Mission Command to Agile Software Development

David P. Harvie
Dr. Arvin Agah

Agenda

- **Introduction**
- Background of Scrum
- Targeted Scrum
- Proposed Experiment
- Initial Observations
- Questions

Problem Statement

- How do you successfully develop working and reliable software, within budget and on time, given ever changing requirements?

Mission Command as Similar Field

- Attempts to bring order to chaotic, complex human endeavors
- Followed similar evolution (industrial-age decomposition to agility)
- Assessments critical to success

Mission Command as Similar Field

Step	Military Decision Making Process	Waterfall Method
1	Receipt of Mission	System Requirements
2	Mission Analysis	Software Requirements
3	COA* Development	Analysis
4	COA Analysis (War Game)	Program Design
5	COA Comparison	Coding
6	COA Approval	Testing
7	Orders Production	Operations

***COA = Course of Action**

Research Hypothesis

Modifications to agile software development based on inspirations from mission command can improve software engineering process in terms of the planning, prioritizing, and communication of software requirements and progress, as well improve the overall software product.

Approach Summary

- Propose to use mission command as inspiration to improve a specific agile software development method (Scrum)
- Addresses two specific weaknesses: lack of initial planning and lack of architecture
- Introduces Product Design Meeting, end state, and Lines of Effort (LOEs)

Agenda

- Introduction
- **Background of Scrum**
- Targeted Scrum
- Proposed Experiment
- Initial Observations
- Questions

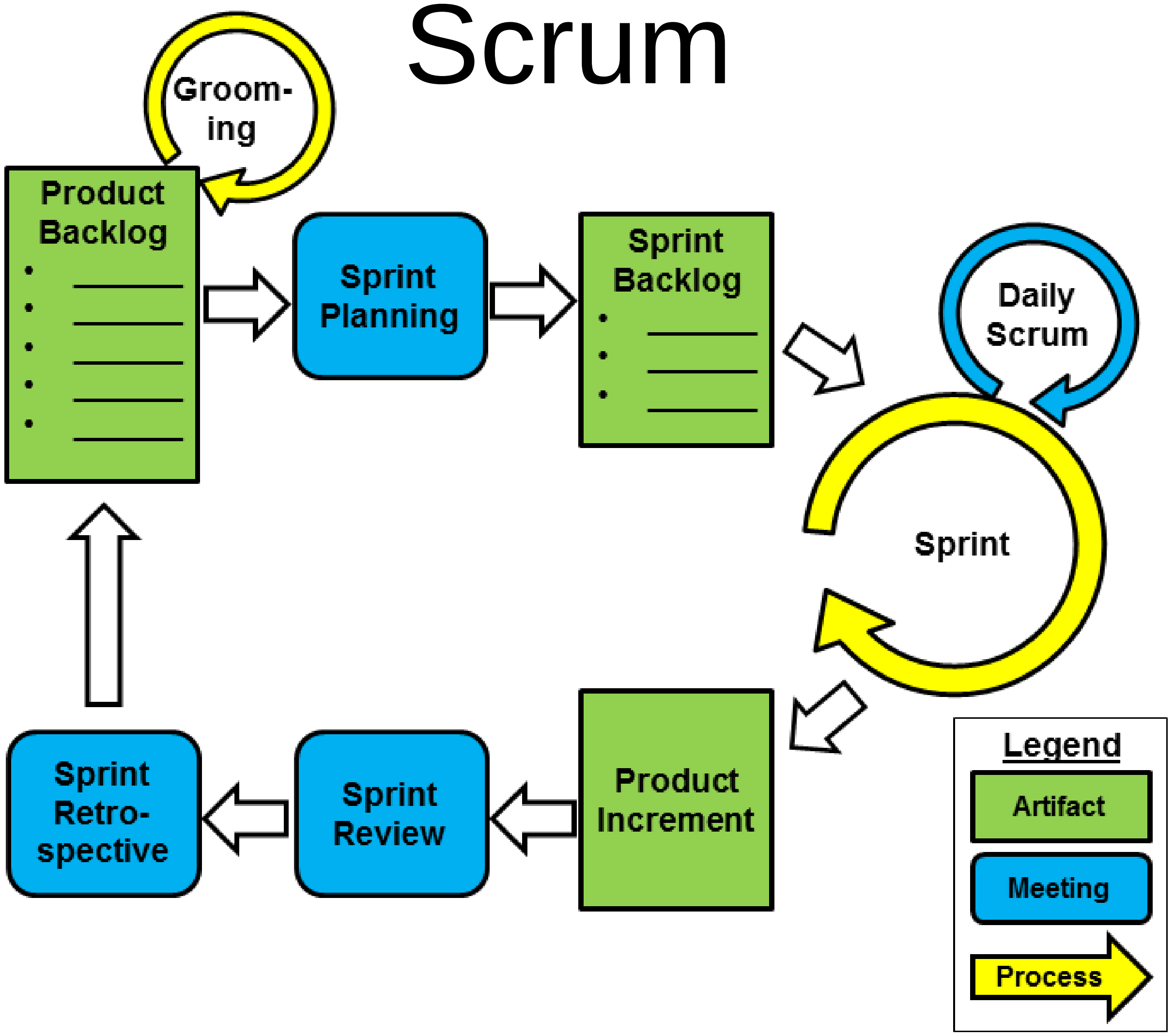
Scrum

- Developed by Ken Schwaber (1996)
- Management process classified as an agile software development method
- Premise: change is unavoidable
- Approach: iteratively build software in increments, conduct continuous assessments, review finished increment

Scrum

- Scrum Team consists of approximately seven people
- Team is cross functional, self-organizing
- Scrum Master – ensures adherence to Scrum principles and removes obstacles
- Product Owner – software client's representative to Scrum Team

Scrum



Groom-ing
Product Backlog
Sprint Planning
Sprint Backlog
Daily Scrum
Sprint
Product Increment
Sprint Review
Sprint Retro-spective
Legend
Artifact
Meeting
Process

Strengths of Scrum

- **Iterative releases** and **feedback** lead to earlier discovery of defects (Sutherland *et al.*, 2007)
- **Continuous feedback** leads to greater software quality (Li *et al.*, 2010)

Weaknesses of Scrum

- **Initial identification of requirements**
  - Creating software before identifying requirements →unnecessary change (Hochmüller & Mittermeir, 2008)
  - Multiple Sprints required before architecture is established (Overhage & Schlauderer, 2012)

Weaknesses of Scrum

- Wrong choice of **Product Owner** hurts team responsiveness (Hochmüller, 2011) (Hoda *et al*., 2010)
- **Lack of design focus** → Planning decisions more tactical than strategic (Drury *et al.*, 2012)

Enhancements of Scrum

- Merge Scrum into more formal methodology such as **CMMI** (Sutherland *et al.*, 2007) (Lina & Dan, 2012) or **PSP** (Rong *et al.*, 2010)
- Develop external framework to manage multiple teams (Maranzato *et al*., 2012)
- Use Line of Balance (LOB) to chart progress (Miranda & Borque, 2010)

Agenda

- Introduction
- Background of Scrum
- **Targeted Scrum**
- Proposed Experiment
- Initial Observations
- Questions

Approach

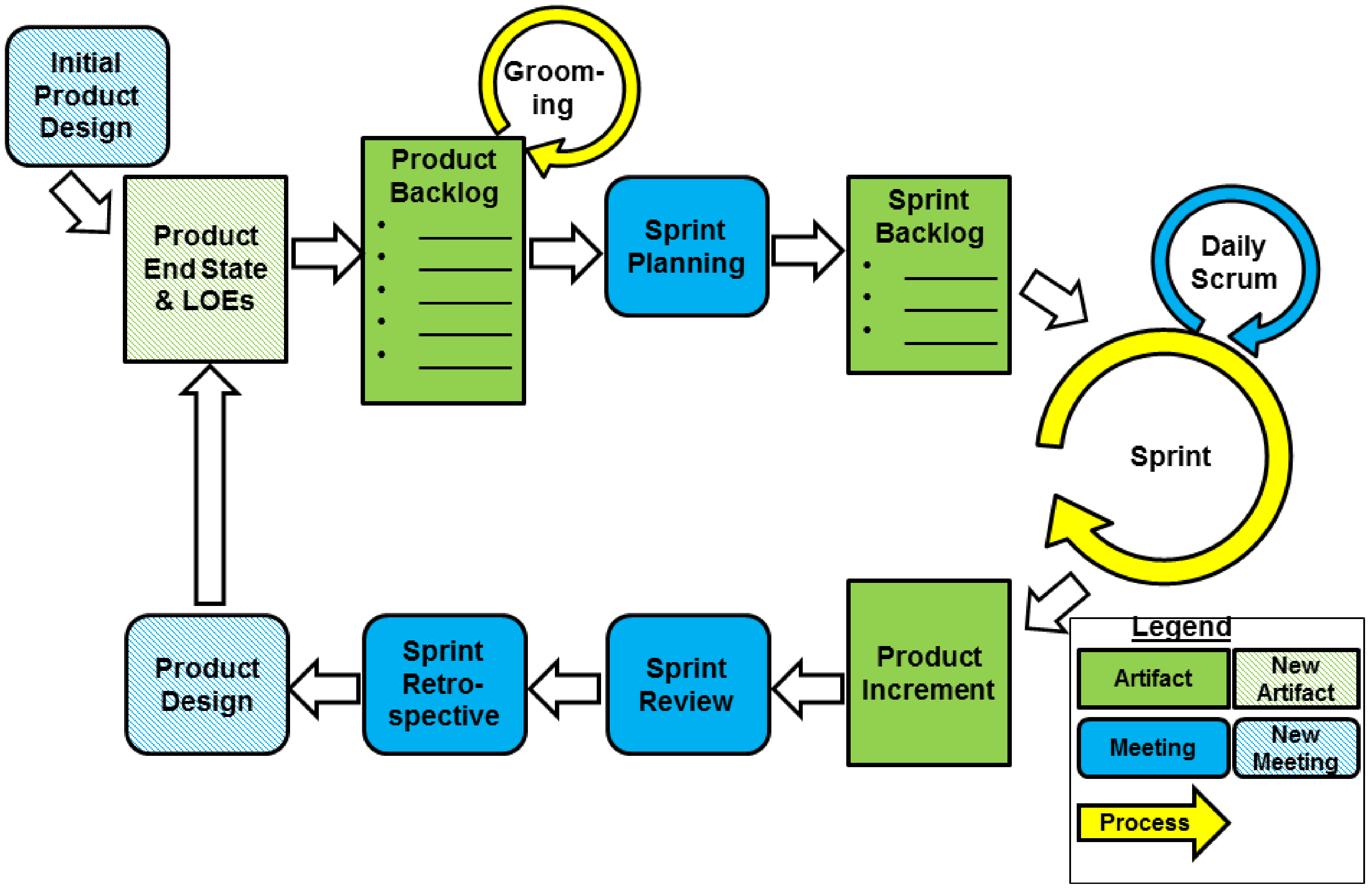
- Specific weaknesses to address:
  - Lack of initial planning and overall design
  - Prioritization of Product Backlog
- Ways to address weaknesses:
  - Addition of Product Design Meeting
  - Use of Lines of Effort (LOEs)

Approach

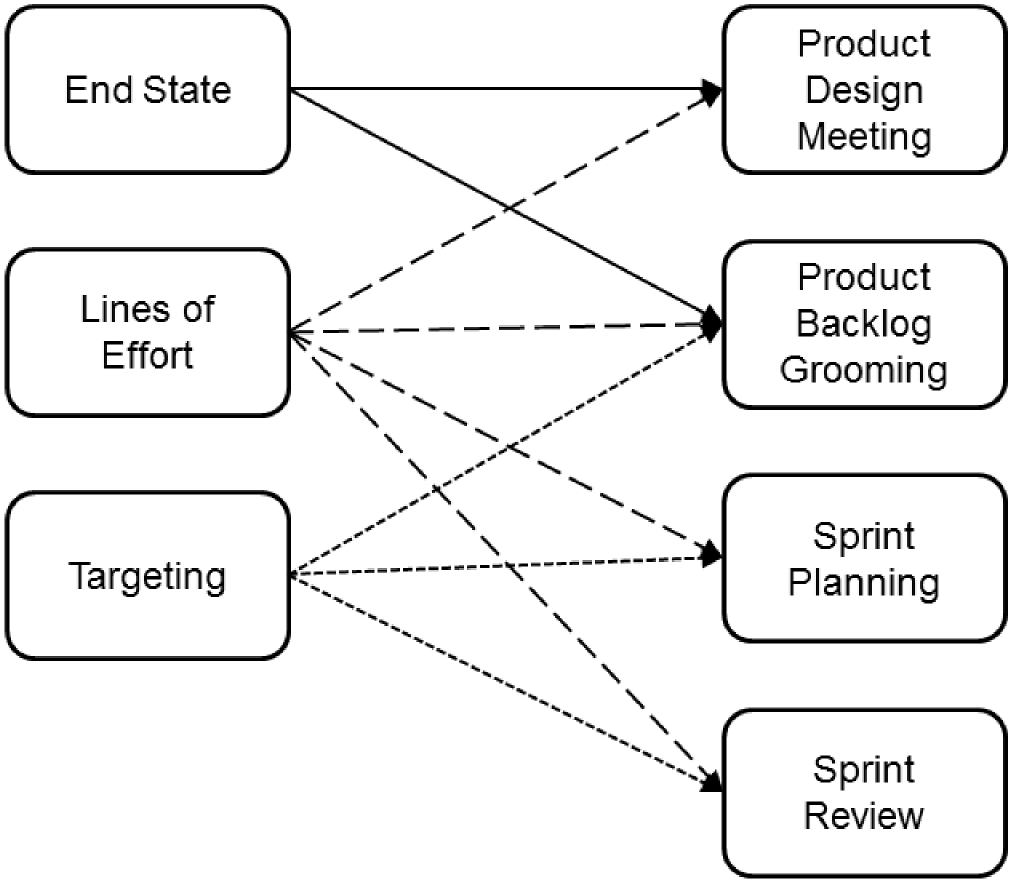
Targeted Scrum will enhance Traditional Scrum by:

1. Product Design Meeting will establish Product framework
2. Product Design Meeting and LOEs will better groom Product Backlog
3. LOEs will improve Sprint Planning
4. LOEs will better communicate progress

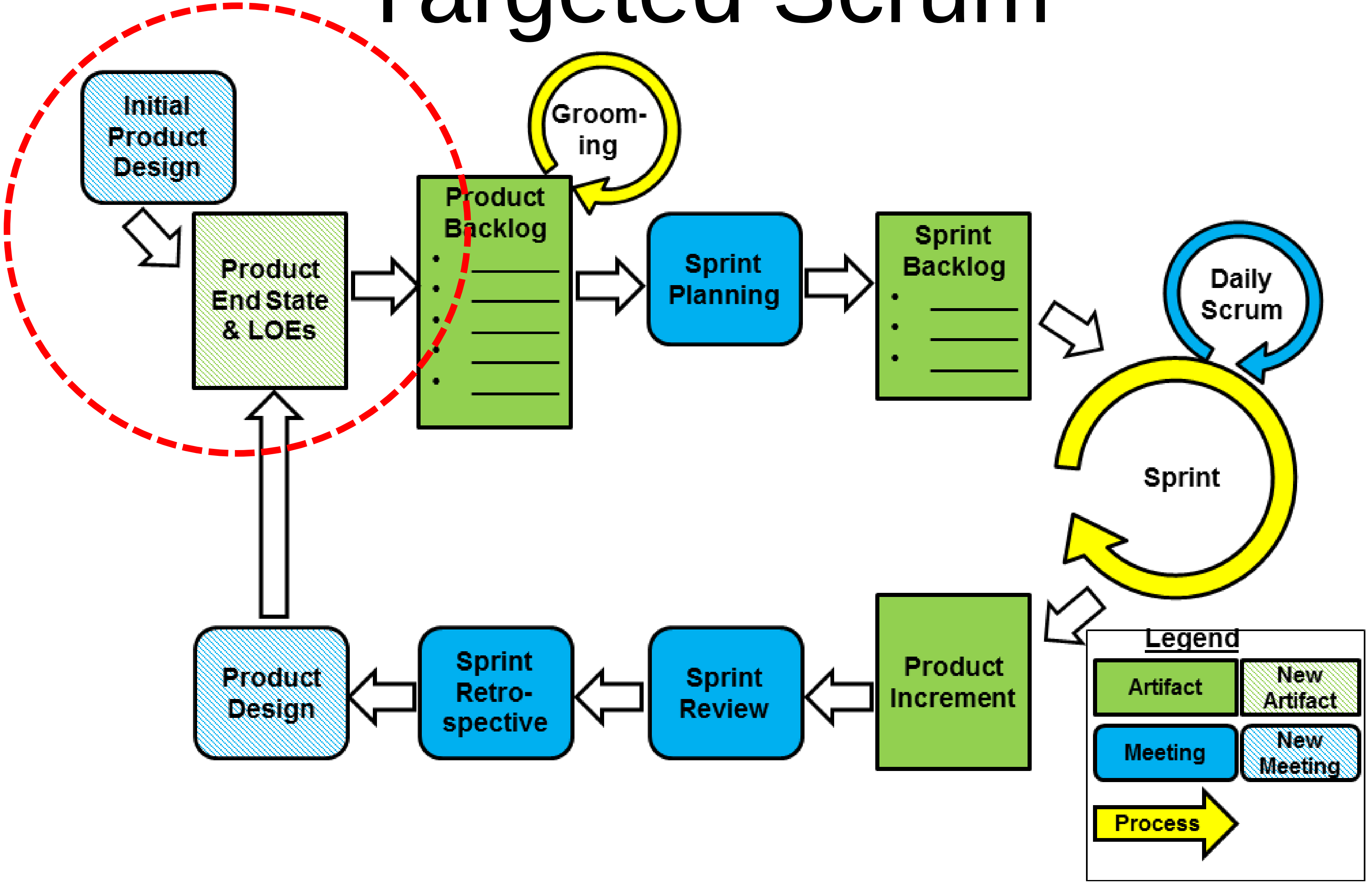
Targeted Scrum



Mapping Mission Command to Modifications



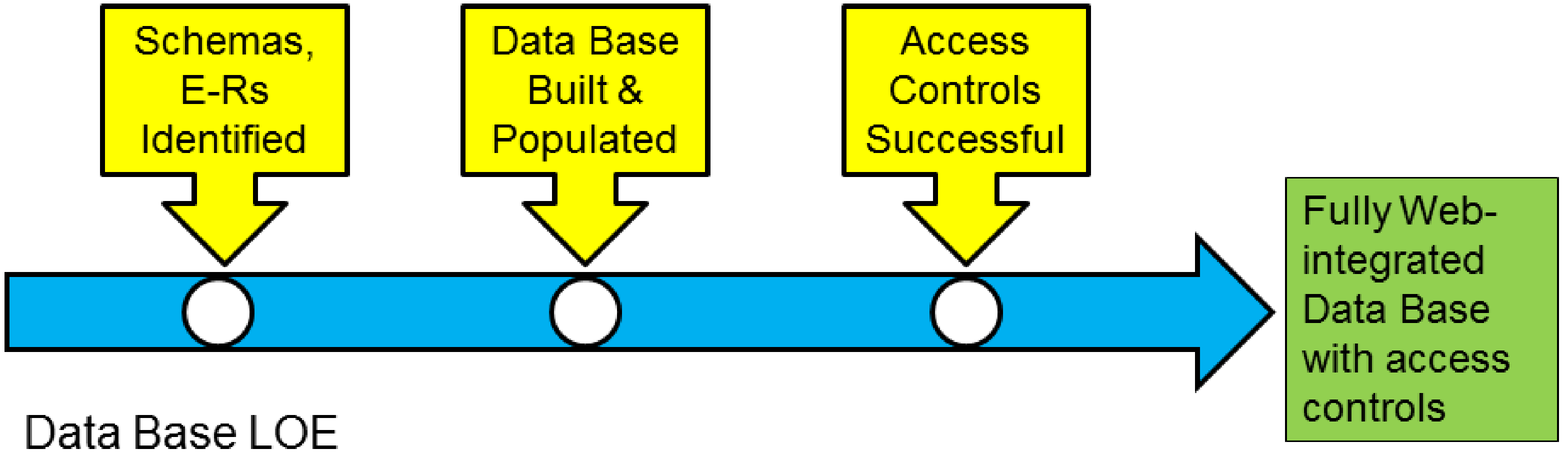
Targeted Scrum



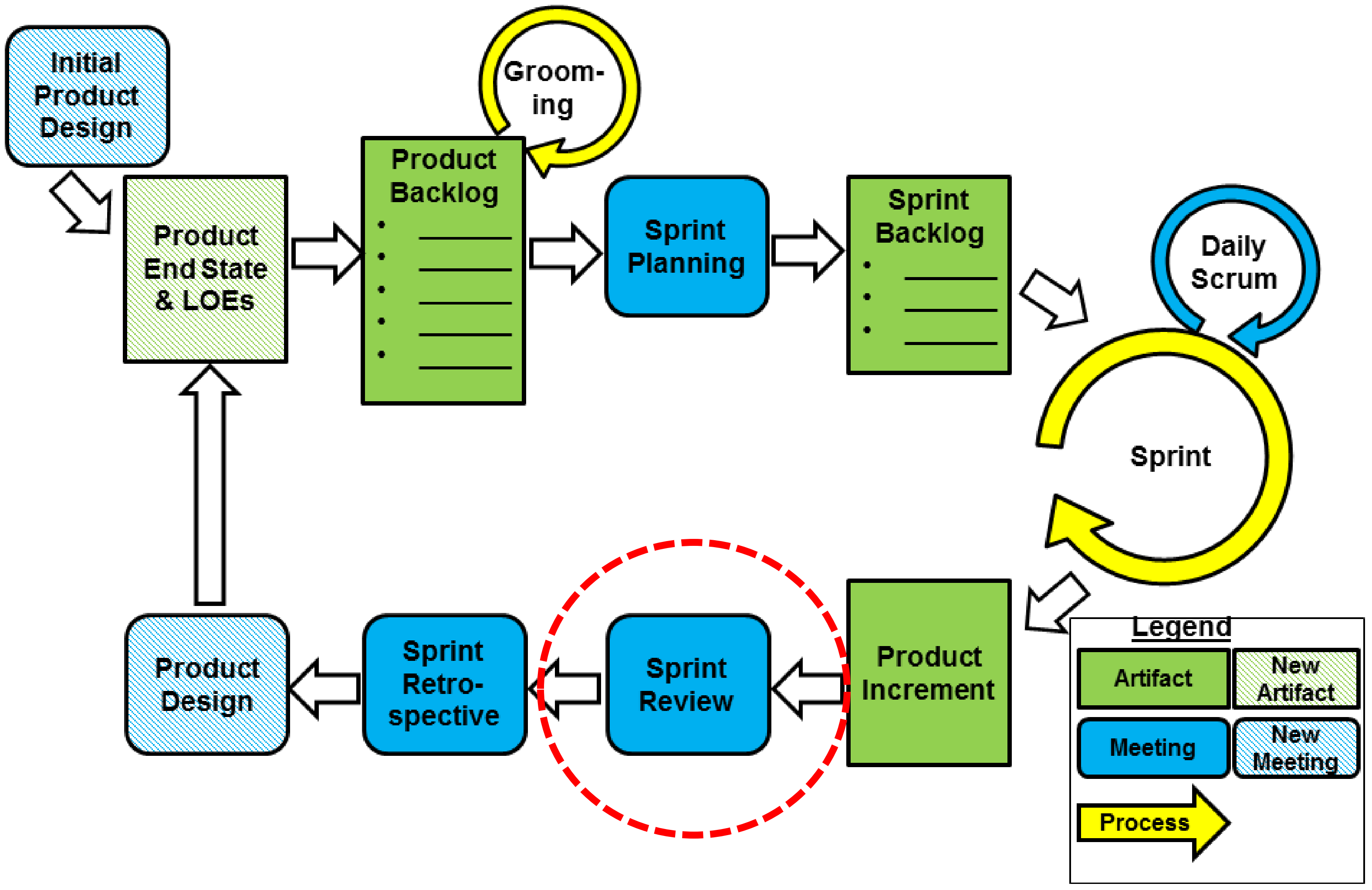
Product End State

A secure, efficient, and easily adaptable personnel management system for an organization of 50-200 people that can be accessed and modified via a Web browser.

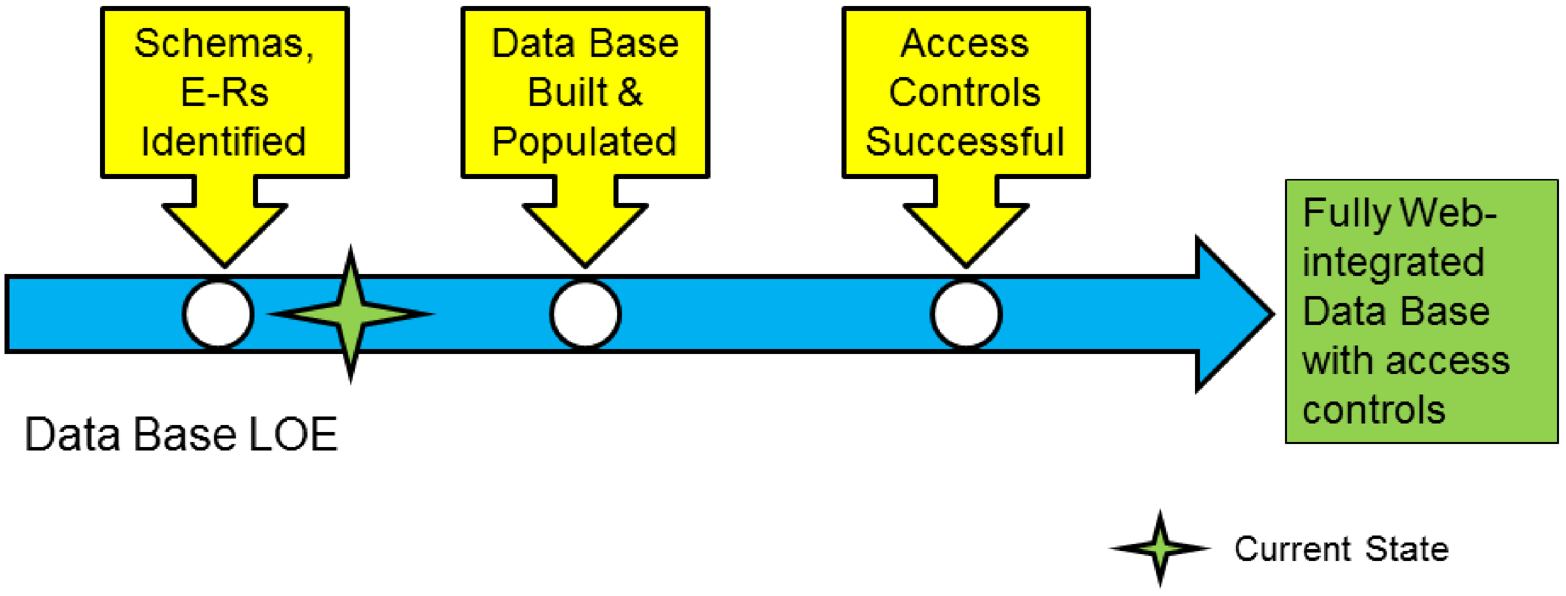
Data Base LOE


Schemas, E-Rs Identified
Data Base Built & Populated
Access Controls Successful
Fully Web-integrated Data Base with access controls
Data Base LOE

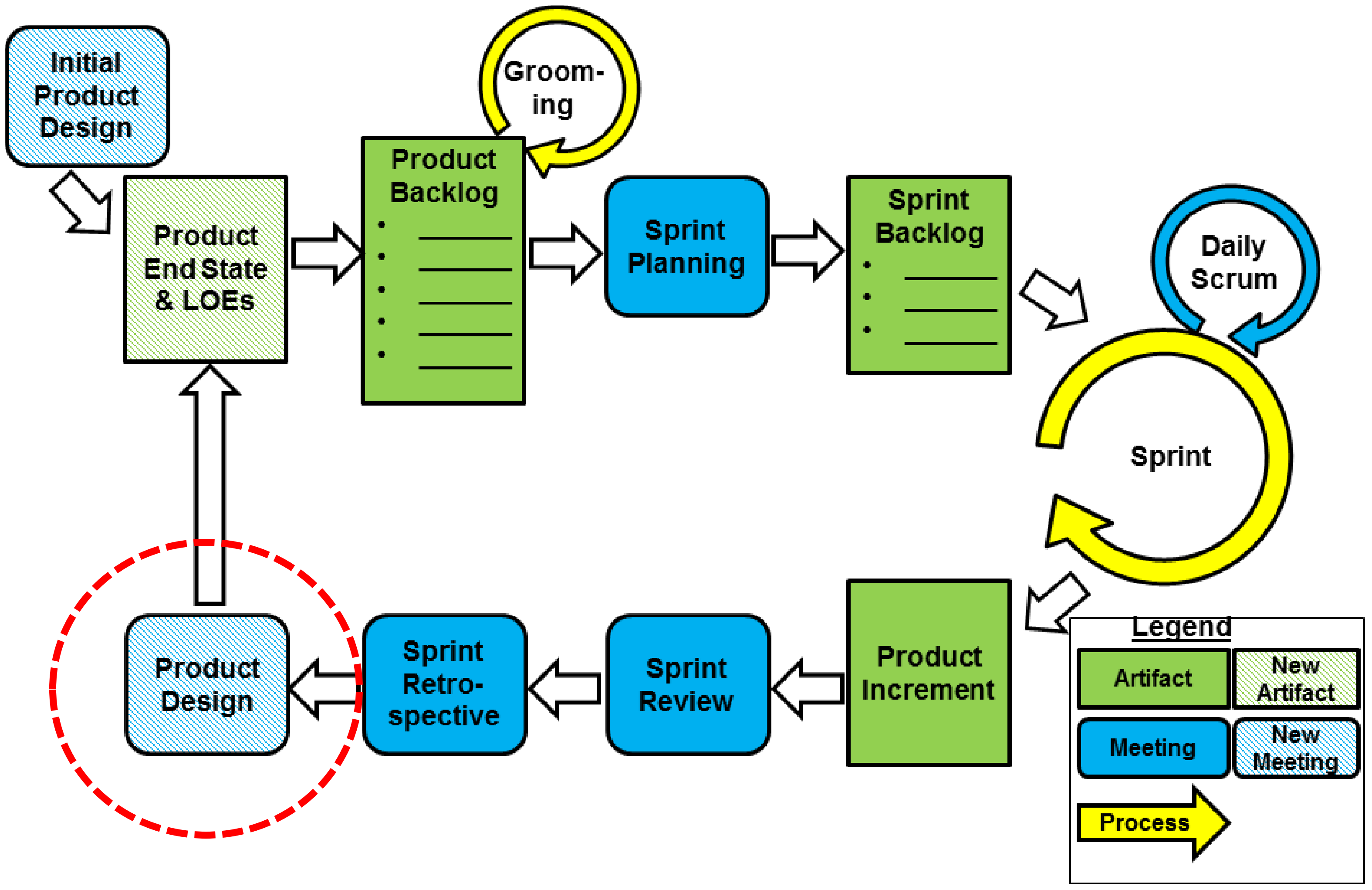
Targeted Scrum



Updated Data Base LOE


Schemas, E-Rs Identified
Data Base Built & Populated
Access Controls Successful
Fully Web-integrated Data Base with access controls
Data Base LOE
Current State

Targeted Scrum



Agenda

- Introduction
- Background of Scrum
- Targeted Scrum
- **Proposed Experiment**
- Initial Observations
- Questions

Protocols

- Develop similar software with both methdologies
- Conduct experiments in Spring 2014
- Set Sprint duration to 2 weeks
- Assign Scrum teams of 3-4 members
- Develop with Java utilizing Eclipse IDE
- Immediately archive all artifacts

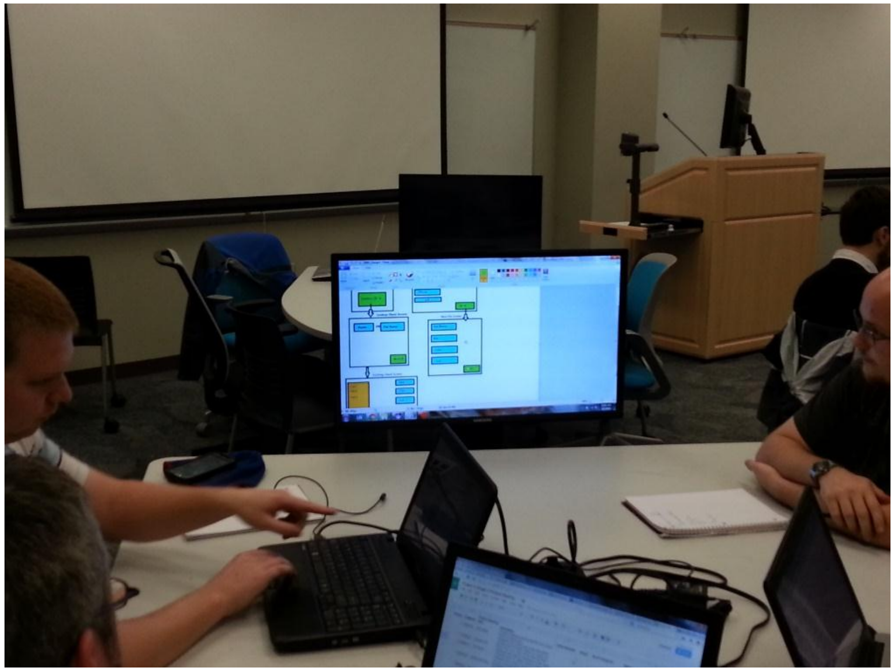
Gathering & Assessing Data

- Each project conducted with split of Traditional/Targeted Scrum teams
- Students surveyed at completion of each project and at end of course
- Outside observers surveyed after the course to focus on artifacts produced
- Product evaluated using automated metrics

Agenda

- Introduction
- Background of Scrum
- Targeted Scrum
- Proposed Experiment
- **Initial Observations**
- Questions

Initial Observations



Initial Observations

- 26 students divided into 7 Scrum teams (junior and senior level undergraduates)
- Student surveys completed at this time
- Utilized model classroom design for new engineering education building

Future Work

- Outside observers assess the artifacts
- Assess the results of the student surveys
- Publish the results and lessons learned from the experiments

QUESTIONS?